ulm university universität
uulm

Institut für Angewandte Informationsverarbeitung
Dr. Andreas F. Borchert

27. Januar 2015
Blatt 13

Objektorientierte Programmierung mit C++ (WS 2014/2015)

Abgabe bis zum 3. Februar 2015, 14:00 Uhr

Lernziele:

- Meta-Programmierung mit rekursiven Templates
- Aufbau von zur Übersetzzeit berechneter Tabellen

Aufgabe 16: Neu-Auflage des Nim-Spiels

In dieser Aufgabe beschäftigen wir uns mit einem etwas verallgemeinerten Nim-Spiel, bei dem in einem Zug von einem Stapel entweder $m_1, m_2, \ldots$ oder m_M Stäbchen entfernt werden können. Der Nim-Wert eines einzelnen Haufens mit der Größe n lässt sich dann rekursiv mit

$$\begin{aligned} \mathcal{G}(0) &= 0 \\ \mathcal{G}(n) &= mex\left(\{\mathcal{G}(n-m_i) \mid n-m_i \geq 0\}\right) \end{aligned}$$

berechnen.[1] Hierbei steht *mex* für eine Funktion, die für eine Menge nicht-negativer ganzer Zahlen die kleinste nicht-negative ganze Zahl liefert, die in der Menge nicht enthalten ist (*minimal exclusive*). Für die leere Menge liefert *mex* entsprechend die 0. $mex(\{0,1,3,4\})$ liefert beispielsweise die 2.

Wenn mehrere Haufen mit $n_1, \ldots, n_N$ Stäbchen vorliegen, dann lässt sich der akkumulierte Nim-Wert mit

$$\mathcal{G}(n_1) \oplus \ldots \oplus \mathcal{G}(n_N)$$

berechnen, wobei $\oplus$ hier die Nim-Addition repräsentiert, die dem bitweisen Exklusiv-Oder entspricht (in C++ wird hierfür der „^“-Operator verwendet).

Ziel der Übungsaufgabe ist es, die Berechnung von $\mathcal{G}$ zur Laufzeit zu vermeiden und stattdessen zur Übersetzzeit eine fertige Tabelle der Werte $\mathcal{G}(0), \ldots, \mathcal{G}(S)$ zu erzeugen, wobei S für die maximale Haufengröße steht. Dann fällt zur Laufzeit die sich wiederholende

[1] Die Berechnung der Nim-Werte wurde dem ersten Band zu *Winning Ways for Your Mathematical Plays* von Elwyn R. Berlekamp, John H. Conway und Richard K. Guy entnommen.

rekursive Berechnung der $\mathcal{G}$ weg, da die benötigten Werte der Tabelle entnommen werden können.

Die Parameter des Nim-Spiels können dann zur Übersetzzeit mit entsprechenden Makrovariablen bestimmt werden:

```
thales$ make MOVES='2, 7, 15, 16' MAX_HEAP_SIZE=30
g++ -g -Wall -std=gnu++11 -DMOVES='2, 7, 15, 16' -DMAX_HEAP_SIZE=30  -c -o RunGame.o RunGame.cpp
g++ -o nim  RunGame.o
thales$ nim
*** Game of Nim ***
Number of heaps: 3
Possible moves: 2 7 15 16
Heaps: 2 6 19
Your move: 0 2
2 items are taken from heap 0
Heaps: 0 6 19
7 items are taken from heap 2
Heaps: 0 6 12
Your move: 2 7
7 items are taken from heap 2
Heaps: 0 6 5
2 items are taken from heap 1
Heaps: 0 4 5
Your move: 1 2
2 items are taken from heap 1
Heaps: 0 2 5
2 items are taken from heap 1
Heaps: 0 0 5
Your move: 2 2
2 items are taken from heap 2
Heaps: 0 0 3
2 items are taken from heap 2
Too bad, you lost!
thales$
```

Hinweise: Die über die Makros definierten Parameter können direkt an die Template-Klasse *NimGame* übergeben werden, die eine variable Zahl von Template-Parametern akzeptiert:

```
typedef NimGame<MAX_HEAP_SIZE, MOVES> Game;
Game game(number_of_heaps);
```

Beginnend mit C++11 werden Template mit einer variablen Zahl von Parametern unterstützt. Hier ist ein Ausschnitt aus *NimGame*, der zeigt, wie das geht:

```
template<unsigned int max_heap_size, unsigned int ... possible_moves>
class NimGame {
    public:
        static constexpr unsigned int moves_len = sizeof...(possible_moves);
        typedef unsigned int moves_type[moves_len];
        static constexpr unsigned int moves[] = {possible_moves...};
        // ...
);
```

Die Template-Klasse *NimGame* erwartet also einen festen Parameter (*max_heap_size* des Typs **unsigned int**) und mit *possible_moves* eine variable Zahl von Parametern. Die gesamte variabel lange Parameterliste kann dann mit *possible_moves*... passend expandiert werden. Wenn wie hier ein **constexpr**-Array erzeugt wird und das später mit nicht konstanten Indizes verwendet wird, ist es notwendig, das Array einmal außerhalb der Template-Klasse passend zu definieren:

```
template<unsigned int max_heap_size, unsigned int ... possible_moves>
constexpr typename NimGame<max_heap_size, possible_moves...>::moves_type
    NimGame<max_heap_size, possible_moves...>::moves;
```

Es gibt eine Vorlage zu dieser Aufgabe, in der bereits einige Teile fertig implementiert sind. In *Set.hpp* ist die fertige Definition der Template-Klasse *IsMemberInSet*, mit der überprüft werden kann, ob eine Zahl in einer Menge enthalten ist. Der erste Template-Parameter ist hier die zu überprüfende Zahl, die weiteren variablen Parameter spezifizieren die Menge. So ist beispielsweise *IsMemberInSet*<2, 0, 1, 2, 3>::*is_member* **true**, da die 2 in $\{0, 1, 2, 3\}$ enthalten ist. Hingegen liefert *IsMemberInSet*<2>::*is_member* den Wert **false**, da die 2 in der leeren Menge nicht enthalten ist.

Darauf aufbauend lässt sich das in der Vorlage nicht enthaltene *Mex.hpp* entwickeln, das die *mex*-Funktion umsetzt. So sollte beispielsweise *Mex*<0, 1, 3, 4>::*value* den Wert 2 liefern.

Ebenso fehlt in der Vorlage *Nim.hpp*, das die Funktion $\mathcal{G}$ umsetzt. So sollte beispielsweise *Nim*<14, 2, 5, 6>::*value* den Wert 1 liefern. Das entspricht $\mathcal{G}(14)$ für $m_1 = 2, m_2 = 5, m_3 = 6$.

In der Vorlage enthalten ist die etwas trickreiche Template-Klasse *Table*, die eine Tabelle konstruiert. Wenn beispielsweise die Template-Klasse *ComputeNimValue* gegeben wird, die für ein n den Wert $\mathcal{G}(n)$ berechnet

```
template <unsigned int N>
struct ComputeNimValue {
    static constexpr unsigned int value = Nim<N, possible_moves...>::value;
};
```

dann lässt sich mit

```
typedef Table<max_heap_size+1, ComputeNimValue> NimValues;
```

eine Tabelle mit *max_heap_size*+1 Werten erzeugen, so dass über *NimValues*::*val*[n] der Wert für $\mathcal{G}(n)$ abgerufen werden kann. Zu beachten ist hier, dass beim zweiten Parameter von *Table* ein nicht instantiiertes Template übergeben wird, das dann von *Table* für die Werte von 0 bis *max_heap_size* instantiiert wird.

Wenn Sie mit der Vorlage loslegen, brauchen Sie nur *Mex.hpp* und *Nim.hpp* zu entwickeln. Es steht Ihnen aber natürlich frei, alles nach eigenen Vorstellungen zu ändern oder zu entwickeln, solange die Tabelle mit den Nim-Werten zur Übersetzzeit berechnet wird.

Sie können Ihre Lösung wieder mit *tar* verpacken und einreichen:

```
thales$ submit cpp 16 nim.tar
```

Viel Erfolg!